Making a Better List-set

EECS 395 “Rust”

Feb. 18, 2016

Linearizability, formally

History H is *linearizable* if it can be extended to complete history G by

- appending responses to some pending invocations, and/or
- discarding the remaining pending invocations

such that there exists an equivalent legal sequential history S where $\rightarrow_G \subseteq \rightarrow_S$.

Example

$$H = \begin{array}{l} \text{A q.enq(3)} \\ \text{B q.enq(4)} \\ \text{B q:void} \\ \text{B q.deq()} \\ \text{B q:4} \\ \text{B q.enq(6)} \end{array}$$

Example

$$H = \begin{array}{l} \text{A q.enq(3)} \\ \text{B q.enq(4)} \\ \text{B q:void} \\ \text{B q.deq()} \\ \text{B q:4} \\ \text{B q.enq(6)} \end{array} \qquad G = \begin{array}{l} \text{A q.enq(3)} \\ \text{B q.enq(4)} \\ \text{B q:void} \\ \text{B q.deq()} \\ \text{B q:4} \\ \\ \text{A q:void} \end{array}$$

Example

$$H = \begin{array}{l} \text{A q.enq(3)} \\ \text{B q.enq(4)} \\ \text{B q:void} \\ \text{B q.deq()} \\ \text{B q:4} \\ \text{B q.enq(6)} \\ \\ \end{array} \qquad G = \begin{array}{l} \text{A q.enq(3)} \\ \text{B q.enq(4)} \\ \text{B q:void} \\ \text{B q.deq()} \\ \text{B q:4} \\ \\ \text{A q:void} \end{array} \qquad S = \begin{array}{l} \text{B q.enq(4)} \\ \text{B q:void} \\ \text{A q.enq(3)} \\ \text{A q:void} \\ \text{B q.deq()} \\ \text{B q:4} \\ \\ \end{array}$$

S is legal and $\sim G$

Can we do better?

Coarse-grained synchronization:

- Lock the whole object for each operation

Can we do better?

Coarse-grained synchronization:

- Lock the whole object for each operation
- Easy to reason about :-)

Can we do better?

Coarse-grained synchronization:

- Lock the whole object for each operation
- Easy to reason about :-)
- But sequential bottleneck :-(

Four strategies

1. Fine-grained synchronization

Four strategies

1. Fine-grained synchronization
    - :-) Can synchronize on different parts of object concurrently

Four strategies

1. Fine-grained synchronization
    - :-) Can synchronize on different parts of object concurrently
    - :-(But lots of locking/unlocking overhead

Four strategies

1. Fine-grained synchronization
   - :-) Can synchronize on different parts of object concurrently
   - :-(But lots of locking/unlocking overhead
2. Optimistic synchronization
   - :-) No need to lock while traversing
   - :-(But need to validate, and may require expensive retries

Four strategies

1. Fine-grained synchronization
   - :-) Can synchronize on different parts of object concurrently
   - :-(But lots of locking/unlocking overhead
2. Optimistic synchronization
   - :-) No need to lock while traversing
   - :-(But need to validate, and may require expensive retries
3. Lazy synchronization
   - :-) Less work needed than optimistic synchronization
   - :-(But contended operations still need to retry

Four strategies

1. Fine-grained synchronization
   - :-) Can synchronize on different parts of object concurrently
   - :-(But lots of locking/unlocking overhead
2. Optimistic synchronization
   - :-) No need to lock while traversing
   - :-(But need to validate, and may require expensive retries
3. Lazy synchronization
   - :-) Less work needed than optimistic synchronization
   - :-(But contended operations still need to retry
4. Lock-free synchronization
   - :-) No longer at the mercy of the scheduler
   - :-(But complex, and maybe high overhead